TP 3 Data Mining: KDE Kernel Density Estimation

Tuesday 26th March, 2024
deadline: Monday 15th April, 2024, 23:59
Optional

Classification using kernels

In this TP you will implement classifications using Kernels which is based in a Naive Bayes classifier where we estimate $p(\mathbf{x}|c)$ for each class c using Bayes theorem to calculate $p(c|\mathbf{x})$.

Kernels

The basic idea in Probability Density Estimation of an unknown density function $p(\mathbf{x})$ is to estimate the probability P of a given region R from a limited number, n, of training examples $\{\mathbf{x}_i | i = 1..n\}$. In theory this probability is given by:

$$P(\mathbf{x} \in R) = \int_R p(\mathbf{x}) d\mathbf{x}$$

For a large number of samples n the expected number of samples k that will fall in the region R will be:

$$E[k] = nP(\mathbf{x} \in R)$$

Now if the region R is small enough we can assume that $p(\mathbf{x})$ does not vary considerably within it, in which case:

$$P(\mathbf{x} \in R) = \int_R p(\mathbf{x}) d\mathbf{x} \simeq p(\mathbf{x}) V_R$$

where V_R is the volume of the region R over which we integrate. So we have:

$$\frac{E[k]}{n} = P(\mathbf{x} \in R) \simeq p(\mathbf{x}) V_R$$

thus we can get an estimate of the $p(\mathbf{x})$ probability density by:

$$p(\mathbf{x}) \simeq \frac{E[k]}{nV_R} \tag{1}$$

Kernel for continuous variables

Gaussian kernel

The univariate of the Gaussian kernel is given by :

$$K(x) = \frac{1}{\sqrt{2\pi}} exp(-\frac{1}{2}x^2) \tag{2}$$

Kernels for discrete variables

- Binary variable

$$K(x, x_i) = \begin{cases} \lambda & x = x_i \\ \\ 1-\lambda & x \neq x_i \end{cases} \tag{3}$$

  It should hold $0.5 < \lambda \leq 1$ so that we have a large K in case of equality and small otherwise.

- Discrete variable with $g > 2$ values

$$K(x, x_i) = \begin{cases} \lambda & x = x_i \\ \\ \frac{1-\lambda}{g-1} & x \neq x_i \end{cases} \tag{4}$$

  where $g - 1 =$ normalization factor so that it holds $\sum_{j=1}^{g} K(.) = 1$.

Multivariate density estimation

General solution: product of univariate kernels. We fit one kernel to each of the d dimensions.

$$\hat{p}(\mathbf{x}) = \frac{1}{n \times h_1 \times \cdots \times h_d} \sum_{i=1}^{n} \prod_{j=1}^{j=d} K_j(\frac{\mathrm{x}_j - \mathrm{x}_{ij}}{h_j}) \tag{5}$$

- If we assume the same kernel and h we can simplify using $k_j = K$, and $h_j = h$ for all dimensions:

$$\hat{p}(\mathbf{x}) = \frac{1}{n \times h^d} \sum_{i=1}^{n} \prod_{j=1}^{j=d} K(\frac{\mathrm{x}_j - \mathrm{x}_{ij}}{h}) \tag{6}$$

Classification using kernels

- Data: TRN = $\{\mathbf{x_i}\}$ for $i = 1, \cdots, n$, $\mathbf{x}$ = instance to classify
- Parameters: function K, size h
- General Algorithm:

```
1. For each $c \in C$
2.     $P(c) \leftarrow \frac{n_c}{c}$
3.     $\text{TRN}_c = \{\mathbf{x_i} | \mathbf{x_i} \in c\}$
4.     $\hat{p}(\mathbf{x}|c) = \frac{1}{n_c h_1 \cdots h_d} \sum_{i=1}^{N_C} \prod_{j=1}^{D} K_j \left( \frac{\mathrm{x}_j - \mathrm{x}_{ij}}{h_j} \right)$
5.     $P(c|\mathbf{x}) \leftarrow \hat{p}(\mathbf{x}|c) P(c)$
6. Return $c$ having maximal $P(c|\mathbf{x})$
```

- Actually Naive Bayes where we estimate $p(\mathbf{x}|c)$ for each class c, and them the Bayes theorem to calculate $p(c|\mathbf{x})$.

Exercises

In this TP, you will first implement appropriate functions for kernel density estimation using Gaussian kernels and study the effect of the bandwidth parameter h using a simple artificial dataset. In the second exercise, you will implement a kernel-based classification algorithm. It looks like to the Naive Bayes classification algorithm, but instead of assuming a normal distribution for the data or using relative frequencies, we estimate the likelihood $p(\mathbf{x}|c)$ for each class c using kernel functions, and then we apply Bayes theorem to compute $p(c|\mathbf{x})$.

1. **First exercise:**
   Estimate the probability density function $p(x)$ from a given set of training points using Gaussian kernels (see Multivariate density estimation assuming the same kernel (Gaussian) and h for all the dimensions).

   The training set consists of the following four points:

c_1	:	1, 1
c_2	:	1, 4
c_3	:	3, 2.5
c_4	:	4, 2.5

   Experiment with the following values of h : 0.3, 0.4, 0.5.

   Create a regular set of points covering the plane $[0, 5] \times [0, 5]$ and store them in the *testSet* variable (provided in tp3_kde_main.ipynb). These points will be input into the density function, allowing us to compute its values regularly.

The training set consists of the points given above and it is stored in variable c.

- Write a function that computes the Gaussian kernel, with a center c_i and a size of hypercube h (bandwidth).
- Plot $p(\mathrm{x}|c_i)$ for every $\mathrm{x} \in testSet$ and every c_i (for h: 0.3, 0.4, 0.5)
- Plot $p(\mathrm{x})$ for every $\mathrm{x} \in testSet$ (for h : 0.3, 0.4, 0.5)

For $h = 0.3$, your graphs should resemble those provided in the following pages. Understand the effect of the bandwidth h on the density estimation.

2. **Second exercise: Classification using Kernel density estimation.**

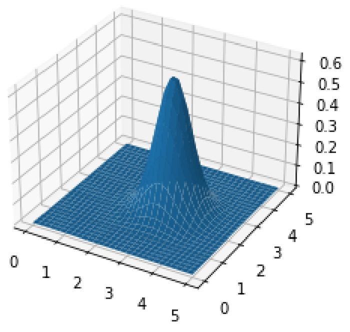
   Implement the kernel-based classification algorithm using the NaiveBayes() class from the first practical work (TP1_NB). Add new functions for density estimation of likelihoods and modify the existing functions of the NaiveBayes() class as necessary.

   For continuous attributes, use the Gaussian kernel, and for discrete attributes, use the corresponding kernel for discrete variables (as described above), depending on whether they are binary or not.

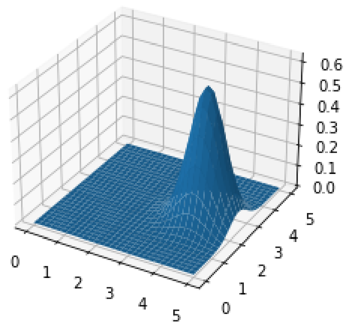
   After completing your implementation,

   - Apply the classifier with kernels to the **Adult dataset**. (As in the first TP_NB, check for missing values, handle them if present, and split the dataset into training and testing sets (80/20) using a random seed.)
   - Compute the test classification accuracy for h**: 0.3, 0.4, 0.5, 1.0**.
   - Discuss the effect of the h parameter on the classification accuracy of the algorithm in detail.

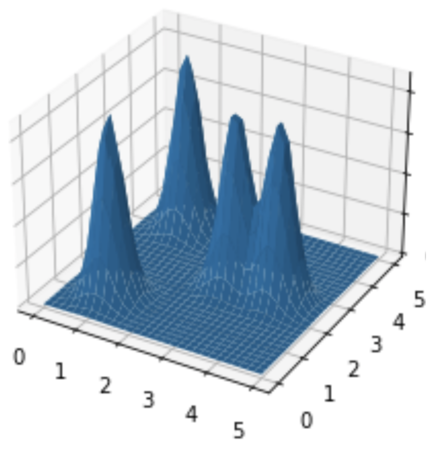
Kernel Function at point : c = [3. 2.5]. h : 0.5



Kernel Function at point : c = [4. 2.5]. h : 0.5



Density Function p(x), h:0.3


0 1 2 3 4 5
0 1 2 3 4 5